



Projeto Interdisciplinar iMed

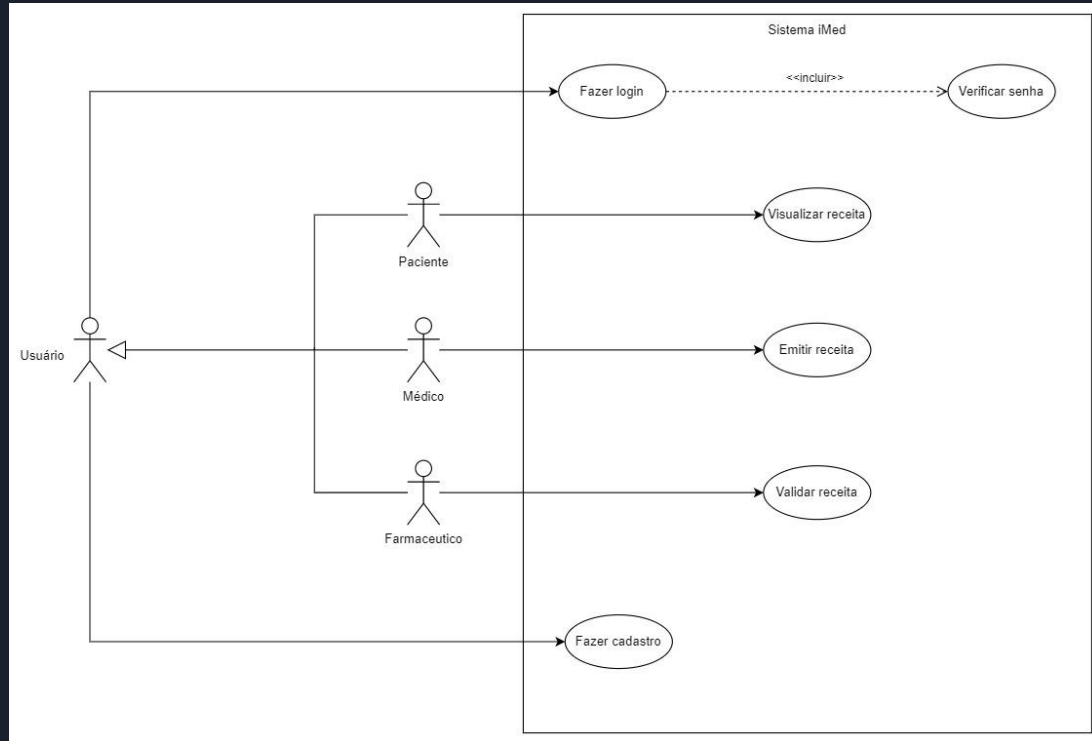
Eduardo Dias Dini
Marco Venicio
Rhuan Martins

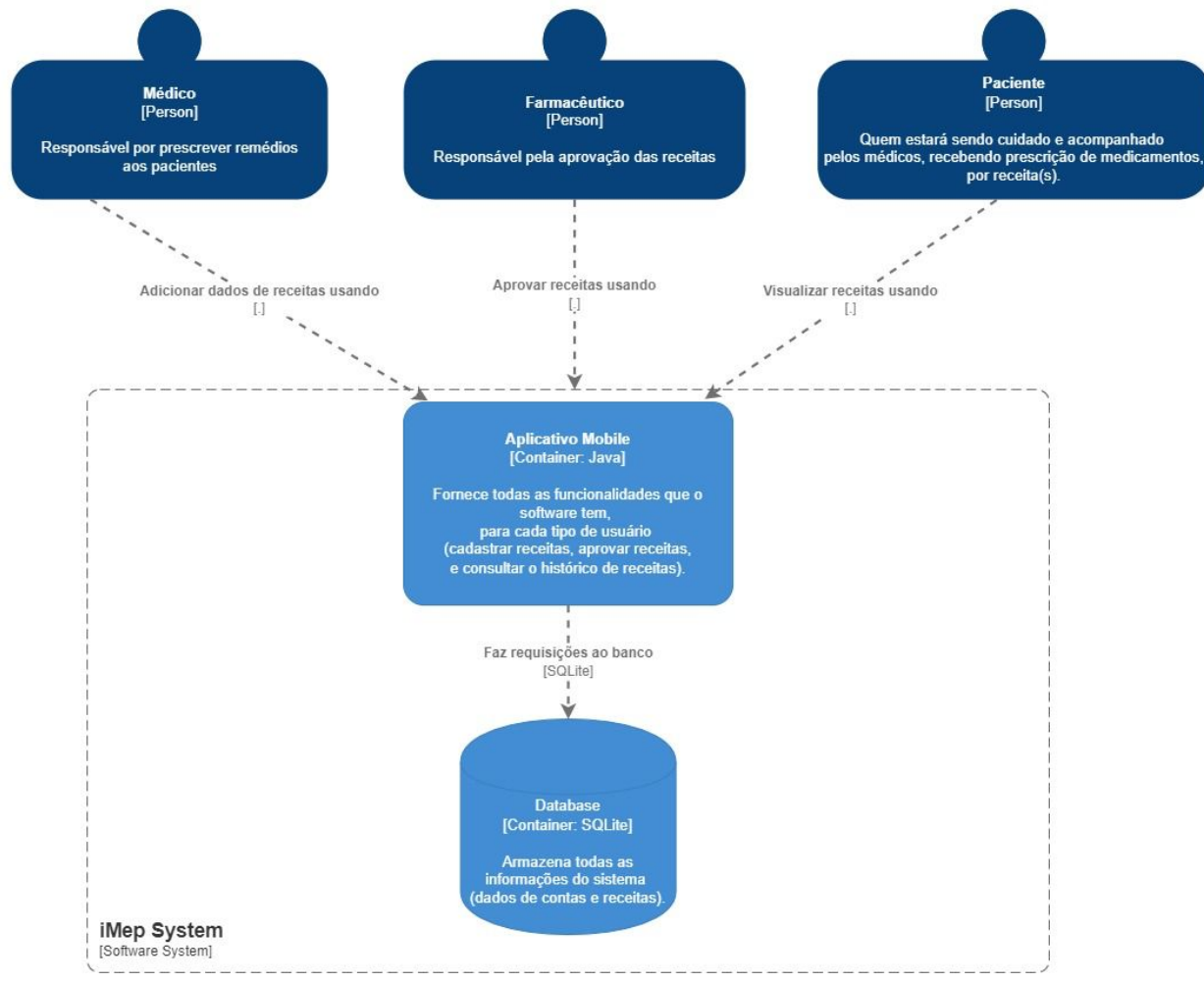


Produto

O objetivo do app iMed é o de agilizar e facilitar a comunicação entre médico, farmacêutico e paciente, além de reduzir custos e evitar fraudes.

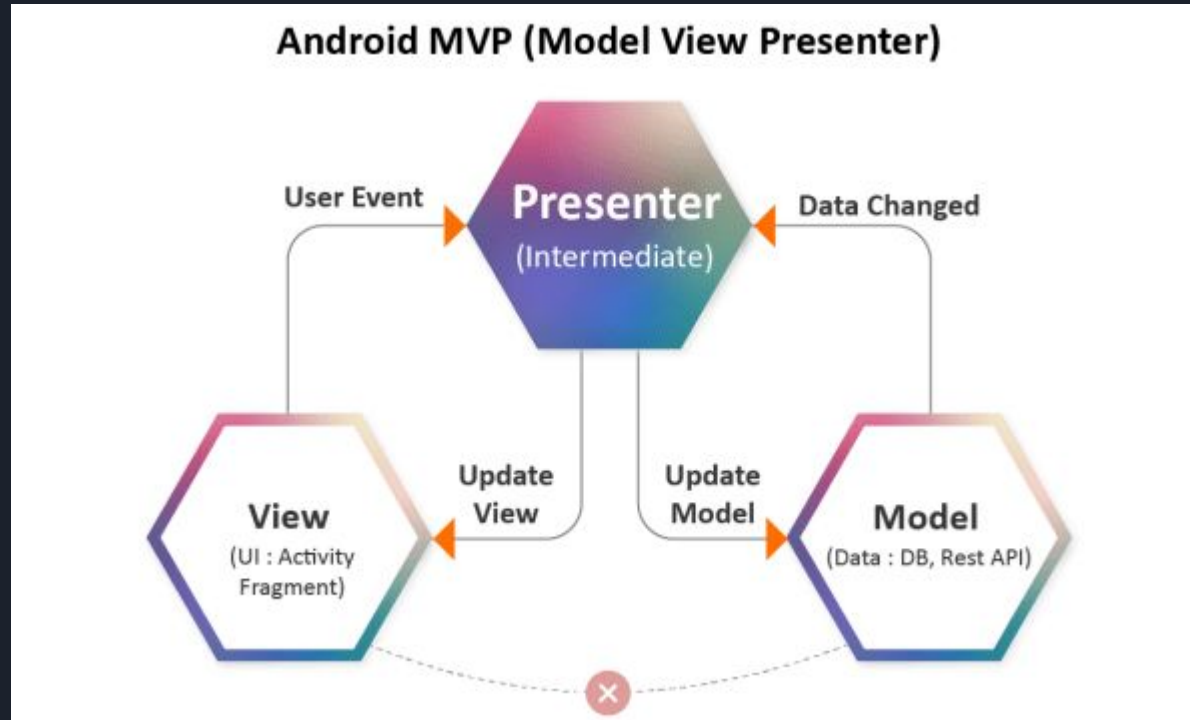
Produto







Arquitetura MVP





Arquitetura MVP

- Não há comunicação entre a *View* e a *Model*;
- Derivação do MVC usado para construção de *user interfaces*;
 - *Model*: define o modelo do dado e o acesso ao banco de dados
 - *View*: é onde os dados são apresentados e guia os comandos do usuário para o *Presenter* atuar sobre os dados;
 - *Presenter*: atua sobre a *Model* e a *View*, recupera os dados do modelo e os formata para exibí-los na *View*;
- Uso de interfaces para diminuir o acoplamento;



Design Pattern - Factory


- Pattern criacional;
- Usado quando todas as classes em potencial estão na mesma hierarquia, encapsula e centraliza a criação de objetos ;
- Fechado para modificação, aberto para extensão.



Design Pattern - Factory

Classe abstrata, encapsulamento, abstração


```
1 package com.example.imed.Model.Usuarios;
2
3 public abstract class Usuario {
4
5     private String nome;
6     private String senha;
7
8     public Usuario() {}
9
10    public String getNome() { return nome; }
13
14    public void setNome(String nome) { this.nome = nome; }
17
18    public String getSenha() { return senha; }
21
22    public void setSenha(String senha) { this.senha = senha; }
25
```



Design Pattern - Factory

Herança

```
1 package com.example.imed.Model.Usuarios;
2
3 public class Paciente extends Usuario {
4
5     private String cpf;
6
7     public Paciente() { super(); }
8
9
10
11     public String getCpf() { return cpf; }
12
13
14
15     public void setCpf(String cpf) { this.cpf = cpf; }
16
17
18
19 }
20
```



Design Pattern - Factory


Polimorfismo

```
1 package com.example.imed.Model.Usuarios;
2
3 public class UsuarioFactory {
4
5     public UsuarioFactory() {}
6
7     @
8     public Usuario criarNovoUsuario(String tipoDeUsuario) {
9
10         switch (tipoDeUsuario) {
11             case "paciente":
12                 return new Paciente();
13             case "medico":
14                 return new Medico();
15             case "farmaceutico":
16                 return new Farmaceutico();
17             default:
18                 return null;
19         }
20     }
21 }
```




Design Pattern - Strategy

- Padrão comportamental;
- Classes semelhantes que diferem apenas no comportamento;
- Encapsula os algoritmos de login dos 3 tipos de usuário;
- Fechado para modificação, aberto para extensão;



Design Pattern - Strategy Interface

```
2  
3 ❶↓ public interface LoginStrategyInterface {  
4  
5 ❷↓     Boolean realizarLogin(String login, String senha);  
6  
7     }
```



Design Pattern - Strategy

Agregação

```
2
3  public class Login {
4
5      private LoginStrategyInterface loginStrategy;
6
7      public Login(LoginStrategyInterface loginStrategy) {
8          this.loginStrategy = loginStrategy;
9      }
10
11     public Boolean realizarLogin(String login, String senha) {
12         return loginStrategy.realizarLogin(login, senha);
13     }
14
15 }
16
```

Design Pattern - Strategy

Implementação da interface

```
7 public class LoginMedico implements LoginStrategyInterface {
8
9     private ClasseDAO dao;
10    private Context context;
11
12    public LoginMedico(Context context) {
13        this.context = context;
14        this.dao = new ClasseDAO(this.context);
15    }
16
17    @Override
18    public Boolean realizarLogin(String login, String senha) {
19        try{
20            if (dao.obterLoginMedico(login).equals(senha)) {
21                return true;
22            } else {
23                return false;
24            }
25        } catch (NullPointerException e){
26            return false;
27        }
28    }
29 }
30 }
```



CRUD

Create

```
34 @ public void inserirPaciente(Paciente paciente){
35     ContentValues values = new ContentValues();
36     values.put("cpf", paciente.getCpf());
37     values.put("paciente_nome", paciente.getNome());
38     values.put("paciente_senha", paciente.getSenha());
39     banco.insertOrThrow( table: "paciente", nullColumnHack: null, values);
40 }
41
```




CRUD

Update (validar receita)

```
170
171     public void inserirFkFarm(String fkFarm, int id){
172         banco.execSQL("UPDATE "+"receita"+" SET fk_farm = "+"'"+fkFarm+"'"+"
173             "WHERE idReceita = "+"'"+id+"'");
174     }
175
```

CRUD

Read

```
110 public Object[] obterReceita(String idReceita){
111     Object[] objeto = new Object[5];
112     String busca = "select idReceita, nome_remedio, horario, dosagem, fk_med from receita where idReceita = '"+ idReceita +"'";
113     Cursor cursor = banco.rawQuery(busca, selectionArgs: null);
114
115     while (cursor.moveToNext()){
116         objeto[0] = cursor.getString(cursor.getColumnIndex(columnName: "idReceita"));
117         objeto[1] = cursor.getString(cursor.getColumnIndex(columnName: "nome_remedio"));
118         objeto[2] = cursor.getString(cursor.getColumnIndex(columnName: "horario"));
119         objeto[3] = cursor.getString(cursor.getColumnIndex(columnName: "dosagem"));
120         objeto[4] = cursor.getString(cursor.getColumnIndex(columnName: "fk_med"));
121     }
122     cursor.close();
123
124     return objeto;
125 }
```



CRUD

```
176 public void deletarContaPaciente(String paciente){  
177     banco.delete( table: "paciente", whereClause: "cpf = ?", new String[]{paciente});  
178 }
```

Fim

